Merge Is Not Recursion

Ramón Casares

ORCID: 0000-0003-4973-3128

This is a review of the book "Why Only Us" by Berwick & Chomsky done from the point of view of computing. The main conclusions are: that the book uses the word 'recursion' with a meaning that does not correspond to the meaning used in mathematics, causing confusion, and that the theory proposed in the book, that syntax is just Merge, cannot be translated sensibly to computing, because Merge by itself cannot even parse, and therefore it is not enough to explain the evolution of Merge to explain the evolution of language, as the book pretends.

Keywords: syntax evolution, Turing completeness, recursion, Merge.

§1 Introduction

¶1 · Being myself an electric engineer, I am an outsider both to linguistics and to evolution, and then I am not optimally suited to amend a book subtitled "Language and Evolution". So, yes, this is yet another review of "Why Only Us" (WOU) by Berwick & Chomsky (2016) (B&C), but this time from a eccentric point of view, computing.

¶2 · I have learned a lot reading WOU, but every time I met the word 'recursion' I felt troubled. This happened surely because my understanding of recursion is different from the one used in linguistics, and therefore I should firstly explain what I think about recursion. I am very sorry, but we will need some mathematics.

§2 Mathematics

§2.1 Recursion

¶1 · Chomsky (2007), page 20, is wrong when he writes: "If the lexicon is reduced to a single element, then unbounded Merge will easily yield arithmetic." The offending word is 'easily'. The definition of Merge, in the same page, is:

$$\text{Merge}(X, Y) = \{X, Y\} .$$

Chomsky (2006), page 184, states it more precisely: "The most restrictive case of Merge applies to a single object, forming a singleton set. Restriction to this case yields the successor function, from which the rest of the theory of natural numbers can be developed in familiar ways." The theory of natural numbers is also known as arithmetic.

¶2 · Let us call the single element of the lexicon $\emptyset$. Then,

$$\text{Merge}(\emptyset) = \text{Merge}(\emptyset, \emptyset) = \{\emptyset, \emptyset\} = \{\emptyset\} ,$$

which is indeed a singleton set. Reiterating we obtain: $\text{Merge}(\text{Merge}(\emptyset)) = \text{Merge}(\{\emptyset\}) = \{\{\emptyset\}\}$, $\text{Merge}(\text{Merge}(\text{Merge}(\emptyset))) = \text{Merge}(\{\{\emptyset\}\}) = \{\{\{\emptyset\}\}\}$, and so on. In the construction of the natural numbers by Zermelo (1908): $0 = \emptyset$, $1 = \{\emptyset\}$, $2 = \{\{\emptyset\}\}$, $3 = \{\{\{\emptyset\}\}\}$, and so on. Now, rewriting, $\text{Merge}(0) = 1$, $\text{Merge}(1) = 2$, $\text{Merge}(2) = 3$, $\text{Merge}(n) = n+1$, and then Merge is indeed the successor function. So far, so good.

¶3 · But the successor function is not enough to develop the theory of natural numbers. This question was answered by Kleene (1952) in his exhaustive investigation of recursion, an investigation that can be traced back to Kleene (1943), and in even to Kleene (1935). Kleene (1952) is the classical text book on recursion, and surely the most complete and instructive text on the subject, so we will use it here.

¶4 · The successor function is only Schema (I), the simplest of the six schemata (I)–(VI) that are needed to define any recursive function of natural numbers, see pages 219 and 279. The six schemata, where x' is the successor of x, and $\mu y R(y)$ is the least (natural number) y such that predicate $R(y)$ holds, are:

(I) successor function:

$$\varphi(x) = x'$$

(II) constant functions:

$$\varphi(x_1, \ldots, x_n) = q$$

(III) identity functions:

$$\varphi(x_1, \ldots, x_n) = x_i$$

(IV) substitution:

$$\varphi(x_1, \ldots, x_n) = \psi(\chi_1(x_1, \ldots, x_n), \ldots, \chi_m(x_1, \ldots, x_n))$$

(V) primitive recursion:

$$\begin{aligned} \varphi(0, x_2, \ldots, x_n) &= \psi(x_2, \ldots, x_n) \\ \varphi(y', x_2, \ldots, x_n) &= \chi(y, \varphi(y, x_2, \ldots, x_n), x_2, \ldots, x_n) \end{aligned}$$

(VI) general recursion:

$$\varphi(x_1, \ldots, x_n) = \mu y[\chi(x_1, \ldots, x_n, y) = 0]$$

¶5 · Using primitive recursions the calculation goes down from y' to y, finishing if it reaches 0, and then using Schemata (I)–(V) we obtain complete functions, but adding general recursion the calculation can go up without bound, and then using Schemata (I)–(VI) we obtain all recursive functions, both complete and partial, as stated by Corollary to Theorem XIX in page 331 of Kleene (1952).

¶6 · The conclusion is that in recursion the successor function is the simplest of the six schemata, and primitive and general recursion play the leading rôles. The six schemata are needed to achieve full recursion, and the successor function is needed because it is the step that applied iteratively without bound can traverse through each and every member of the infinite enumerable set of natural numbers, so it is the essential step to effectively enumerate the natural numbers.

§2.2 Lambda-calculus

¶1 · As the successor function is a restriction of Merge, we could think that the lack of power of the successor function is caused by the restriction, but that Merge is much more powerful. We will see that that is not the case investigating another formalism: lambda-calculus. Lambda-calculus is mathematically equivalent to recursion, as proved by Kleene (1936). The classical text books on lambda-calculus are Curry & Feys (1958) and Barendregt (1985).

¶2 · The syntax of lambda-calculus can be defined inductively this way:

$$x \in V \qquad V \subset \Lambda$$
$$x \in V \Rightarrow x' \in V$$
$$x \in V,\ M \in \Lambda \Rightarrow (\lambda x\, M) \in \Lambda$$
$$M, N \in \Lambda \Rightarrow (M\, N) \in \Lambda$$

This means that in lambda-calculus there are only: an infinity of variables built by priming, which belong to the subset V; lambda abstractions, that is, function definitions, $(\lambda x\, M)$; and function applications, $(M\, N)$. These last two are ordered pairs that grow into binary trees by induction.

¶3 · Applications of Merge also grow into binary trees by induction, with the only difference that in lambda-calculus order matters. We will call the version of Merge used in lambda-calculus `cons`, a name taken from Lisp, see McCarthy (1960). So `cons` takes two objects and returns the ordered pair containing the two objects, while Merge takes two objects and returns the two-element set containing the two objects.

$$\texttt{cons}(X, Y) = (X, Y)$$

Order is the only difference between Merge and `cons`.

¶4 · In order to allay any doubt, we will define a version of lambda-calculus that uses Merge instead of `cons`, and then two-element sets instead of ordered pairs.

$$x \in V \qquad V \subset \Lambda$$
$$x \in V \Rightarrow x' \in V$$
$$x \in V,\ M \in \Lambda \Rightarrow \{\lambda x, M\} \in \Lambda$$
$$M, N \in \Lambda \Rightarrow \{\mu M, N\} \in \Lambda$$

We have needed an additional terminal, μ, to tag the function in function applications, so the mu-tagged element is the function, and the non-tagged element is the argument to apply. We can call it lambda-mu-calculus, and it is, except on its syntax, identical to lambda-calculus.

¶5 · In lambda-calculus, `cons` is the operation that builds composed data structures, just as Merge is in lambda-mu-calculus and in language. But we have only presented the syntax of lambda-calculus, that is, the enumeration of its well-formed formulas, also known as lambda-expressions. And lambda-calculus is a calculus because there is a way to transform a lambda-expression into another lambda-expression, a way called beta-reduction by Curry & Feys (1958). So beta-reduction is to lambda-calculus, as the set of substitution, primitive and general recursion is to recursion. In other words, in lambda-calculus, the rôle of substitution and recursions is played by beta-reduction. Following the analogy, the set of the natural numbers is infinite enumerable and it can be traversed using the successor function, the set of lambda-expressions is infinite enumerable and it can be traversed using the `cons` operation, and the set of lambda-mu-expressions is infinite enumerable and it can be traversed using the Merge operation.

§2.3 Computing

¶1 · The only form of recursion mentioned in WOU is computing, and then, though it is widely known, it demands an introduction, at least a short one. The classic reference of computing is its founding paper by Turing (1936), where he presents his *a*-machine, now deservedly known as Turing machine, and his universal computing machine, now known as universal Turing machine. A universal Turing machine takes the description of any Turing machine M and any data D as input data, and returns what the Turing machine M returns when it takes data D as input. In other words, a universal Turing machine is a Turing machine that can be programmed to emulate any Turing machine.

¶2 · Computing is mathematically equivalent to recursion, and to lambda-calculus, as proved by Turing (1937), who uses some results already proved by Kleene (1936). The equivalence means that for each recursive function, which is any obtained using the six schemata, there is (at least) a Turing machine that performs the same calculation, and the converse, and similarly for lambda-defined functions.

¶3 · All forms of recursion deal with infinite enumerable sets, and then they require some form of infinity, but in computing it is possible to separate cleanly the computing capacity from the infinities. Thus, a Turing machine has not time limitations, and it is composed by a finite state computing module attached to an infinite tape, which is just external memory. This way computing abstracts away the limitations of a device in speed and memory from its computing capacity. A universal Turing machine is a Turing machine, and therefore it is also a finite state computing module attached to an infinite tape. We say that the computing capacity of the finite state computing module of a universal Turing machine is Turing complete. Now, it is important to realize i) that it is possible to build physically a Turing complete device, and the CPU of a full programmable computer is an example, and ii) that a Turing complete device has the computing power of a universal Turing machine, though it can meet some time and memory limitations that the universal Turing machine will not meet.

¶4 · This is a good moment to explain how I understand recursion and equivalent formalisms, as lambda-calculus or computing. In any of them there is a base set that is

infinite enumerable, and the key is that, assuming Church's thesis, a specified finite set of operations allows the definition of any effective calculation on the members of the base set. The base set is the set of natural numbers in recursion, the set of lambda-expressions in lambda-calculus, and the set of strings of symbols in computing. Of these three, only lambda-expressions are hierarchical, because both natural numbers and strings are linear.

§3 Review

¶1 · Now I ask you to read WOU as if you were me, that is, using the concepts seen in §2. In particular, I beg you please to be specially aware of the word 'recursion', and remember its meaning, as explained in §2, every time it is mentioned.

§3.1 Uncontroversial, page 9

¶1 · After stating that language is basically hierarchical and unbounded, and not linear, B&C write in page 9 (emphasis in the original): "Not only do hierarchical properties rule the roost in human syntax, they have no real upper bound, [...]. If one subscribes to the Church-Turing thesis along with the assumption that the brain is finite, then there is no way out: we *require* some notion of recursion to adequately describe such phenomena. So much is uncontroversial."

¶2 · I subscribe to the Church-Turing thesis, to the finiteness of the brain, and to the hierarchical and unbounded nature of syntax, and yet I had to write my first question mark near the word *'require'* in my new and pristine copy of WOU.

¶3 · To me, recursion is *required* to allow any possible effective calculation on an infinite enumerable set, be it hierarchical or linear, but I do not think they were meaning that, because they were insisting in the hierarchical and non-linear nature of language. On the other hand, I was only in page 9.

§3.2 Iterated without bound, page 70

¶1 · In page 70, B&C define "the simplest mode of recursive generation: an operation that takes two objects already constructed, call them X and Y, and forms from them a new object that consists of the two unchanged, hence simply the set with X and Y as members. We call this optimal operation Merge. Provided with conceptual atoms of the lexicon, the operation Merge, iterated without bound, yields an infinity of digital, hierarchically structured expressions." This deserves two notes.

¶2 · First: Merge is a simple operation, I concede, but it is not perhaps the simplest composing operation, or at least some argument should be provided to convince me. For example, it seems that for McCarthy (1960), `cons` is simpler than Merge, given that he implemented `cons` in LISP, but not Merge.

¶3 · Second: Syntax generation needs, in addition to Merge, an iterating operation to yield an infinity of expressions. Merge by itself does not iterate. As it is explained in the quoted text, Merge simply takes two objects and returns the two-element set that contains the two objects. So Merge is just a step. If Merge, or its result, needs some kind of iteration, then some other operation is needed. It is not Merge that yields infinities, it is iteration without bound.

§3.3 Optimally, page 71

¶1 · Defending the Strong Minimalist Thesis, B&C write in page 71: "Optimally, recursion can be reduced to Merge." My understanding of recursion is not compatible with that sentence, see §2. Or they are using 'recursion' to mean something related to hierarchical that I cannot precise, or they are ignoring, at least, an iterating operation that can be applied to Merge in order to generate sentences that are longer than two words, as explained in the second note of §3.2.

¶2 · Recursion cannot be reduced to Merge, because Merge by itself cannot calculate every recursive function, and in particular it cannot parse. To calculate every recursive function, at least another operation playing the rôle that beta-reduction plays in lambda-calculus, see §2.2, is needed. That operation could be Unification, proposed by Jackendoff (2011). That Merge by itself cannot even parse is shown for example by Ginsburg (2016)[1], who needs nine operations and rules complementing Merge to parse.

§3.4 Protolanguage, page 72

¶1 · I disagree twice with B&C in page 72.

¶2 · B&C argue that there were no protolanguage, because or i) there is Merge, and then sentences are not bounded, so there is full language, or ii) there is not Merge, and then there are not sentences, so there is no language. I see it otherwise: i) if there is Merge, and no iteration but only Merge, then all sentences will be two-word sentences, as Progovac (2016) suggests, and ii) without Merge a protolanguage without syntax, or with one-word sentences, is possible, as Bickerton (2014) suggests. And these two possibilities are not mutually exclusive.

¶3 · The second disagreement in page 72 is not important, but let us see it because it is repeated twice. B&C write: "Given two syntactic objects X, Y, Merge can construct a larger expression in only two logically possible ways: either X and Y are disjoint; or else one is a part of the other." Why that word 'logically'? In set theory it is also possible that sets X and Y have some elements in common, and some other elements that the other set has not, for example $X = \{\text{two}, \text{words}\}$ and $Y = \{\text{many}, \text{words}\}$. So, perhaps 'syntactically' there are only two ways, though this is not explained, but as it is, the statement is false. This 'logical' error is repeated in pages 99 and 128.

§3.5 All computers implement Merge, page 98

¶1 · B&C write in page 98: "Every computational system has embedded within it somewhere an operation that applies to two objects X and Y already formed, and constructs from them a new object Z. Call this operation Merge." This is plainly false.

¶2 · A Turing machine, which is the prototype of computational system, does not do any Merge. The only operation of the Turing machine is an 'if' operation: if the current state is q_i, and the read symbol is s_r (which can be *blank*), then transition to state q_n (which can be equal to q_i or not), write symbol s_w (which can be equal to s_r or not, even *blank*), and move to the *left*, or to the *right*, or *halt*.

¶3 · Two other examples are a NOT gate and a binary on-off switch. Though simple, they do computations, so they are computational systems, and they do not Merge anything.

[1] Kindly pointed to me by Berwick.

§3.6 Merge generates infinities, page 112

¶1 · In page 112 there is a summary, so B&C are repeating some ideas about Merge and recursion that bother me. They redefine Merge again, and add: "Merge can then apply recursively to this new hierarchically structured syntactic object, yielding, for example, *the guy read books*. In this way, Merge builds an infinite array of hierarchically structured representations."

¶2 · Comparing this quote to that in page 70, see §3.2, it seems that now 'recursively' means 'iteratively without bound'. Again, the fact that Merge itself cannot build infinities, nor do syntax parsing, is hidden behind imprecise wording.

§3.7 Gallistel problem, page 131

¶1 · Arguing against the idea of an evolutionary progression of cognitive capacities, B&C write in page 131: "It seems that insect navigation [...] requires the ability to 'read from' and 'write to' simple tape-like memory cells. But if so, that's all one needs for a Turing machine."

¶2 · Read from and write to external memory is needed for a Turing machine, but in addition to these interfaces an 'if' operation is needed to determine what to write given the symbol that was read, see the details in §3.5.

¶3 · On the other hand, that every computing system can be modeled as a Turing machine does not imply that all are equal in computing power, the same way that one cannot reason that because a binary on-off switch is implemented with digital electronics, and a full programmable computer is also implemented with digital electronics, then both have the same computing power.

¶4 · Only universal Turing machines can be programmed to execute any computable behavior, and not every Turing machine is a universal Turing machine. There is then a very important distinction between a non-programmable basic calculator, with its fixed set of operations, and a full programmable computer, which is Turing complete because it has the computing capacity of a universal Turing machine. For example, any full programmable computer, even one that has not embedded within it a Merge operation, as it is the case of a universal Turing machine, can be programmed to do Merge, see §3.5. In the case of non-programmable computing systems, most cannot do Merge, though you can implement one that can only do Merge.

¶5 · Therefore, if, for example, the insect cannot be instructed (or programmed) to do arithmetic, as we can be, then it has not "climbed all the way up Nature's ladder" (page 132). For an alternative view on cognitive progression, see my Casares (2016a).

§3.8 Any computation at all, page 134

¶1 · The reader should be ready to understand what bothers me when I read the following sentence in page 134: "Consider a Turing machine, which after all ought to be able to carry out any computation at all."

¶2 · To summarize. A specific Turing machine can carry out one specific computation. For any specific computation, there is a specific Turing machine that can carry it out. A specific universal Turing machine can be programmed to carry out any computation at all, but not every Turing machine is a universal Turing machine. Part of the trick is to understand that the specific universal Turing machine is also carrying out *one* specific computation, that of emulating any specific Turing machine, which is given to the universal one as input data (the program).

¶3 · Therefore, the sentence can be corrected just by adding one word, universal, thus: "Consider a universal Turing machine, which after all ought to be able to carry out any computation at all."

§4 Discussion

§4.1 Unfair

¶1 · My review in section §3 only points to what sounds bad to me, giving a very negative impression of WOU. This is unfair. I have enjoyed reading WOU, and I agree with most of what is written in it.

¶2 · This is a non-exhaustive list of my agreements with what B&C have written in WOU: Universal Grammar is the object of language evolution, syntax is hierarchical, position in syntax hierarchy has meaning, syntax hierarchy is unbounded, randomness plays a rôle in evolution, selection cannot generate novelty, chimps are quite unlike us linguistically, language evolved for thought rather than for communication, there is only one language, our species is probably the first with a recursive language, language did not evolve in the last 60k years, and more.

§4.2 Turing completeness

¶1 · But there is one thing in WOU that I do not buy: that Merge is recursion. From the above, I guess that my understanding of recursion is not what is meant by recursion in WOU, and perhaps in linguistics generally. But, after reading WOU, I still think that using 'recursion' to mean 'discrete and hierarchically unbounded', or 'iterated without bound', is misleading. So I would suggest to do finer distinctions on the computing capacity of devices, and particularly I would urge anybody interested in the evolution of language to discriminate Turing completeness from general computing, reserving 'recursion' for 'Turing completeness'.

¶2 · Take for example the statement in page 126: "'Counting' languages are not natural languages." That there are not counting natural languages seems to imply that Universal Grammar cannot count. But we can count. This would mean that the evolution of Universal Grammar could not explain why we can count. Therefore we would have to assume that there is some Universal Calculator that explains our counting, or arithmetic, capacity, and that it evolved separately from Universal Grammar. As no other species seems to count, this would ask for two very improbable and unique capabilities to evolve separately. And note that arithmetic was the reason why recursion was investigated in the first place, see §2.1.

¶3 · A solution to this counting problem is to assume that Universal Grammar can count, because it is recursive, but that during the critical period some restrictions are imposed in order to make language acquisition easier. I am meaning 'recursive' in the sense of §2, so to be clearer, I am saying that Universal Grammar is Turing complete.

¶4 · Of course, this solution suggests that the evolution of language could not be exactly as presented in WOU; see my own ideas on the evolution of syntax in Casares (2016b). And this solution also explains the fact that we can learn counting artificial languages, asking us to distinguish first language *acquisition* from later languages *learning*.

§4.3 Words

¶1 · One of the more surprising mathematical properties of recursion is self reference. In any kind of recursion, be it proper recursion, lambda-calculus, computing, or else, an element of the base set can refer to any object of the recursive system. For example, Gödel (1930) was able to assign to any arithmetic expression, or sequence of expressions, a unique natural number, so he could write: "Thus we have a proposition before us which asserts its own unprovability", where the proposition is more or less equivalent to the English sentence 'this sentence is false'.

¶2 · This is interesting by itself, but what is important here is that recursive self reference applied to language requires that any word could be a referent, that is, that any word could be referred to, and this transcends the word as a mere reference. Then, it seems to me that the transition from the word as a property of an object, and therefore always referring to the object, to the word as an object itself, and therefore autonomous and not necessarily referential, can be just a consequence of the "recursion-ization" of our brain. And this applies also to sentences and texts, that achieve also a full object status by which abstract definitions and unreal worlds are possible.

¶3 · What I am saying is that self reference is a property of any recursive system, and then, once Turing completeness is reached, self reference comes automatically for free. And therefore when our Universal Grammar achieved Turing completeness, our language could at once refer not only to external objects but also to its own objects, thus extending our world enormously. Language exceeds reality.

§4.4 Merge

¶1 · B&C affirm that Merge is the simplest composing operation, but I think that `cons` is simpler than Merge. When order is prescribed by syntax, parsing is simpler and needs less memory, because then the parser always knows what to expect. On the other hand, I was convinced by B&C that order does not matter internally and that order is a requirement of the vocal interface. The question is then: Why did evolution choose Merge instead of `cons`? It could be just by chance, as B&C argue convincingly in WOU. Or perhaps Merge, being more flexible in generation, was selected. Or Merge was repurposed and its original task demanded no order. This could be the case if Merge was used by vision, where the input image is converted into a hierarchical 3-D model, according to Marr (1982), and also acknowledged in WOU page 136 though not listed in page 158. But this are just possibilities on which I cannot decide.

¶2 · Let us recapitulate what we have seen of Merge. Merge is not recursion, because Merge by itself cannot calculate every recursive function, see §2, and there are versions of recursion that do not use Merge, as for example computing, see §3.5.

¶3 · In addition, Merge by itself is of little use. If the two-element set created by Merge cannot be inspected nor transformed, then it is useless. Even adding substitution, Schema (IV) of recursion by which the result of a Merge can be applied to another Merge, see §2.1, the only use of Merge with substitution would be to create always bigger data structures, wasting memory. The conclusion is that Merge is useless, and even detrimental, except if it is complemented with other operations that can take advantage of the data structures generated by Merge, along with some rules that trigger and control its use. The computational system presented by Ginsburg (2016), already mentioned in §3.3, which requires ten high level operations and rules (including Merge) to parse, is a good example that

Merge by itself, without a carefully designed set of complementary operations, cannot parse. So Merge is just like a single part of a clockwork. In summary, Merge by itself makes little computational sense.

¶4 · On the other hand, Merge can be used to implement recursion, as for example the version of lambda-calculus that we have called lambda-mu-calculus in §2.2.

§5 Conclusion

¶1 · Language evolution requires a multidisciplinary approach, because it is a complex subject touching different problems, and all of them have to be considered simultaneously in order to achieve satisfactory solutions. In an environment like this, it is mandatory to share a common vocabulary, and it would be advisable to share a common theory. Ideally, the theory should be simple, or otherwise it would be difficult to translate it to every discipline involved. But, as Einstein said: "Everything should be made as simple as possible, but no simpler."

¶2 · I have found WOU lacking in both: vocabulary and theory. WOU uses the word 'recursion' with a meaning that does not correspond to the meaning used in mathematics, which is the discipline where the word was born. That is unfortunate, because mathematical recursion is a far reaching concept that has a very precise meaning, while recursion in WOU is a vague concept that blurs important computational distinctions damaging WOU's theory of the evolution of language.

¶3 · B&C theory in WOU starts with what they call the Basic Property of language, namely, that language has three parts: a syntax core and two interfaces to interact with two other systems, the conceptual and the sensorimotor systems (page 11). The core is a finite computational system that generates hierarchical and unbounded structures, and then it *requires* recursion (page 9). Assuming the Strong Minimalist Thesis, recursion can be reduced to Merge (page 71), and Merge is as simple as logically possible (page 111).

¶4 · After reducing syntax to Merge, B&C can explain the evolution of language. Merge is so simple that it could be the result of a minor mutation (page 70) that enhanced the conceptual system (page 101) and was luckily preserved by evolution (page 25). At this stage, Merge should be considered part of the conceptual system. Later, a more complex interface with the sensorimotor system was enabled, by evolution or rather by the plasticity of the brain, so that language could be used as a communication tool (page 108). Only then can we construe language as a third system that provides a new link between the conceptual and the sensorimotor systems.

¶5 · Thus, the evolutionary theory of language in WOU depends exclusively on Merge, and consequently on Merge's recursivity and computational power. But Merge is not recursion, and Merge cannot even parse, so syntax cannot be reduced to Merge. Merge by itself makes a very limited computational system, though Merge can be part of a Turing complete one, that is, part of a computational system that implements recursion in the mathematical sense. But then it is not enough to explain the evolution of Merge to explain the evolution of language, as WOU pretends.

¶6 · In summary, the theory of language evolution proposed in WOU is too simple; it is so simple that it cannot be translated sensibly to computing, because the theory in WOU reduces syntax to Merge, and Merge by itself has not enough computational power to parse and even less to implement mathematical recursion. Therefore, in my opinion, WOU fails from a computational point of view.

¶7 · But, I repeat, it would be unfair to keep just that verdict, and forget all the many valuable points that B&C have made in WOU. I sincerely hope that this review of WOU can be used constructively to enhance the theory of the evolution of language.

References

Barendregt (1985): Henk P. Barendregt, *The Lambda Calculus, its Syntax and Semantics*; Revised Edition, Studies in Logic and the Foundations of Mathematics, Vol. 103, North-Holland Publishing Co, Amsterdam, 1985, ISBN: 0-444-87508-5.

Berwick & Chomsky (2016): Robert C. Berwick, and Noam Chomsky, *Why Only Us: Language and Evolution*; The MIT Press, Cambridge MA, 2016, ISBN: 978-0-262-03424-1.

Bickerton (2014): Derek Bickerton, *More than Nature Needs: Language, Mind, and Evolution*; Harvard University Press, Cambridge MA, 2014, ISBN: 978-0-674-72490-7.

Casares (2016a): Ramón Casares, "Problem Theory"; arXiv:arXiv:1412.1044.

Casares (2016b): Ramón Casares, "Syntax Evolution: Problems and Recursion"; arXiv:1508.03040.

Chomsky (2006): Noam Chomsky, *Language and Mind*, Third Edition; Cambridge University Press, Cambridge, 2006, ISBN: 978-0-521-67493-5.

Chomsky (2007): Noam Chomsky, "Of Minds and Language"; in *Biolinguistics*, vol. 1, pp. 9–27, 2007, URL: http://www.biolinguistics.eu/index.php/biolinguistics/article/view/19.

Curry & Feys (1958): Haskell B. Curry, and Robert Feys, with William Craig, *Combinatory Logic*, Vol. I; North-Holland, Amsterdam, 1958, ISBN: 978-0-7204-2207-8.

Davis (1965): Martin Davis (editor), *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*; Dover, Mineola, New York, 2004, ISBN: 978-0-486-43228-1. Corrected republication of the same title by Raven, Hewlett, New York, 1965.

Ginsburg (2016): Jason Ginsburg, "Modeling of problems of projection: A non-countercyclic approach"; in *Glossa*, vol. 1, no. 1, art. 7, pp. 1–46, 2016, DOI: 10.5334/gjgl.22.

Gödel (1930): Kurt Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I"; in *Monatshefte für Mathematik und Physik*, vol. 38, pp. 173–198, 1931, DOI: 10.1007/BF01700692. Received November 17, 1930. English translation in Davis (1965).

Jackendoff (2011): Ray Jackendoff, "What Is the Human Language Faculty? Two Views"; in *Language*, vol. 87, no. 3, pp. 586–624, September 2011, DOI: 10.1353/lan.2011.0063.

Kleene (1935): Stephen Kleene, "General Recursive Functions of Natural Numbers"; in *Mathematische Annalen*, vol. 112, no. 1, pp. 727–742, December 1936, DOI: 10.1007/BF01565439. Presented to the American Mathematical Society, September 1935.

Kleene (1936): Stephen Kleene, "λ-Definability and Recursiveness"; in *Duke Mathematical Journal*, vol. 2, pp. 340–353, 1936, DOI: 10.1215/s0012-7094-36-00227-2.

Kleene (1943): Stephen Kleene, "Recursive Predicates and Quantifiers"; in *Transactions of the American Mathematical Society*, vol. 53, no. 1, pp. 41–73, January 1943, DOI: 10.2307/1990131.

Kleene (1952): Stephen Kleene, *Introduction to Meta-Mathematics*; Ishi Press, New York, 2009, ISBN: 978-0-923891-57-2. Reprint of the same title by North-Holland, Amsterdam, 1952.

Marr (1982): David Marr, *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*; W.H. Freeman and Company, San Francisco, CA, 1982, ISBN: 0-7167-1284-9.

McCarthy (1960): John McCarthy, "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I"; in *Communications of the ACM*, vol. 3, no. 4, pp. 184–195, April 1960, DOI: 10.1145/367177.367199.

Progovac (2016): Ljiljana Progovac, "A Gradualist Scenario for Language Evolution: Precise Linguistic Reconstruction of Early Human (and Neandertal) Grammars"; in *Frontiers in Psychology*, vol. 7, art. 1714, 2016, DOI: 10.3389/fpsyg.2016.01714.

Turing (1936): Alan Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem"; in *Proceedings of the London Mathematical Society*, vol. s2-42, no. 1, pp. 230–265, 1937, DOI: 10.1112/plms/s2-42.1.230. Received 28 May, 1936. Read 12 November, 1936.

Turing (1937): Alan Turing, "Computability and λ-Definability"; in *The Journal of Symbolic Logic*, vol. 2, no. 4, pp. 153–163, December 1937, DOI: 10.2307/2268280.

Zermelo (1908): Ernst Zermelo, "Untersuchungen über die Grundlagen der Mengenlehre I", DOI: 10.1007/978-3-540-79384-7_6; in *Ernst Zermelo Collected Works* Volume I, H.-D. Ebbinghaus, A. Kanamori (editors), pp. 160–229, Springer-Verlag, Berlin Heidelberg, 2010, ISBN: 978-3-540-79383-0.